Branch: master

Find file | Copy path

Practical-Deep-Learning-For-Coders / 02b_SOTA.ipynb

muellerzr Add files via upload

9befa7d on Sep 13, 2019

1 contributor

Download | History

2.35 MB

Lesson 2, State of the Art and Testing Computer Vision Models

	Tuesday 4-5:15pm	**Friday 4-5:30pm**
Week 1	Introduction	Introduction
Week 2	Custom computer vision tasks	**State of the art in Computer Vision**
Week 3	Introduction to Tabular modeling and pandas	Pandas workshop and feature engineering
Week 4	Tabular and Image Regression	Feature importance and advanced feature engineering
Week 5	Natural Language Processing	State of the art in NLP
Week 6	Segmentation and Kaggle	Audio
Week 7	Computer vision from scratch	NLP from scratch
Week 8	Callbacks	Optimizers
Week 9	Generative Adversarial Networks	Research time / presentations
Week 10	Putting models into production	Putting models into production

- Key items for this week:
  - How do we test for State-Of-The-Art
  - What do we look for?
  - Introductiong to "pieces"

In [0]:
```
!pip install fastai --upgrade
```

In [0]:
```
from fastai.vision import *
```

Dataset:

Our dataset today will be ImageWoof. Link (https://github.com/fastai/imagenette)

Goal: Using no pre-trained weights, see how well of accuracy we can get in x epochs

This dataset is generally harder than imagenette, both are a subset of ImageNet.

Models are leaning more towards being faster, more effecient

In [0]:
```
def get_data(size, woof, bs, workers=None):
    if   size<=128: path = URLs.IMAGEWOOF if woof else URLs.IMAGENETTE
    elif size<=224: path = URLs.IMAGEWOOF_320 if woof else URLs.IMAGENETTE_320
    else          : path = URLs.IMAGEWOOF if woof else URLs.IMAGENETTE
    path = untar_data(path)

    n_gpus = num_distrib() or 1
    if workers is None: workers = min(8, num_cpus()//n_gpus)

    return (ImageList.from_folder(path).split_by_folder(valid='val')
            .label_from_folder().transform(([flip_lr(p=0.5)], []), size=size)
            .databunch(bs=bs, num_workers=workers)
            .presize(size, scale=(0.35,1))
            .normalize(imagenet_stats))
```

In [0]:
```
data = get_data(128, True, 64)
```

Key Ideas:

- Label Smoothing Cross Entropy - State of the art in classification loss functions (We will explore this more in week 6) paper (https://arxiv.org/abs/1906.11567)
  - Basically threshold instead of 1/0
- MixUp docs (https://docs.fast.ai/callbacks.mixup.html)
  - Mixup involves "mixing" sets of images together instead of feeding raw images
- Optimizer
- Activation Functions

If you stil do not understand these do not worry, we will go over them in more detail week 6. Today is about giving you the toys to play with, and understand what a push for SOTA looks like

We will be following a progression that started on the fastai forums here (https://forums.fast.ai/t/meet-mish-new-activation-function-possible-successor-to-relu/53299/) on August 26th of this year.

In this "competition" included:

in this "competition" included:

- Less (https://forums.fast.ai/u/lessw2020)
- Seb (https://forums.fast.ai/u/seb)
- Mikhail Grankin (https://forums.fast.ai/u/grankin)
- Federico Lois (https://forums.fast.ai/u/redknight)
- Ignacio Oguiza (https://forums.fast.ai/u/oguiza)

Label Smoothing Cross Entropy

I won't go into specifics of how it all works, as that will be for week 6 . However here is the code:

In [0]:
```
from torch.distributions.beta import Beta

def lin_comb(a, b, frac_a): return (frac_a * a) + (1 - frac_a) * b

def unsqueeze(input, dims):
    for dim in listify(dims): input = torch.unsqueeze(input, dim)
    return input

def reduce_loss(loss, reduction='mean'):
    return loss.mean() if reduction=='mean' else loss.sum() if reduction=='sum' else loss

class LabelSmoothingCrossEntropy(nn.Module):
    def __init__(self, ε:float=0.1, reduction='mean'):
        super().__init__()
        self.ε,self.reduction = ε,reduction

    def forward(self, output, target):
        c = output.size()[-1]
        log_preds = F.log_softmax(output, dim=-1)
        loss = reduce_loss(-log_preds.sum(dim=-1), self.reduction)
        nll = F.nll_loss(log_preds, target, reduction=self.reduction)
        return lin_comb(loss/c, nll, self.ε)
```

Mixup:

A very basic example:

`new_image = t * image1 + (1-t) * image2`

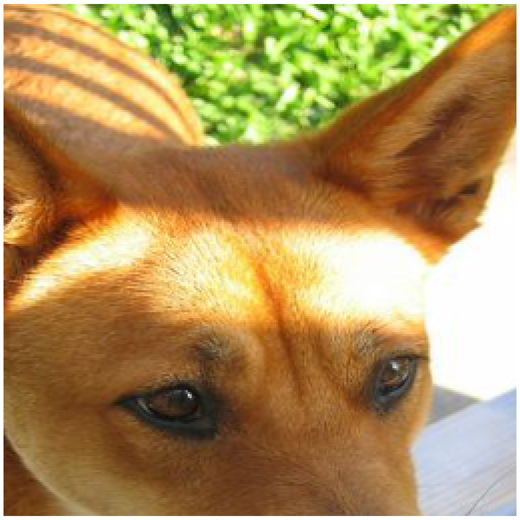
Where `t` is a float between 0 and 1. The target we assign is the same combination as the original, `new_target = t * target1 + (1-t) * target2`

In [0]:
```
img1, lbl1 = data.train_ds[0]; lbl1
```

Out[0]: `Category n02115641`

In [0]:
```
img1
```

Out[0]:



In [0]:
```
img2, lbl2 = data.train_ds[1]; lbl2
```

Out[0]: `Category n02115641`

In [0]:
```
img2
```

Out[0]:

In [0]:
```
t = 0.4
```

In [0]:
```
new_image = t * img1.data + (1-t) * img2.data
new_target = t * lbl1.data + (1-t) * lbl2.data
```

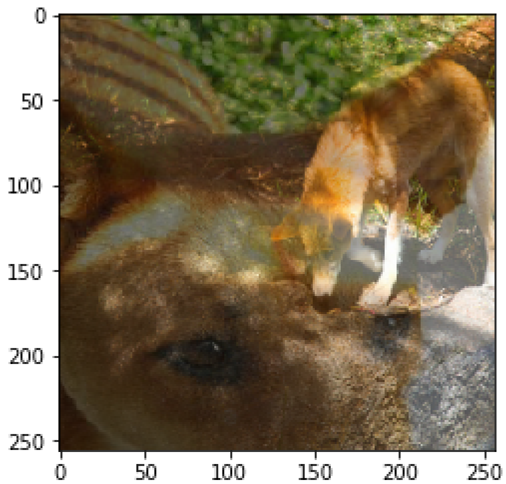
In [0]:
```
im = image2np(new_image)
```

In [0]:
```
im = image2np(new_image)
```

In [0]:
```
import matplotlib.pyplot as plt
```

In [0]:
```
plt.imshow(im)
```

Out[0]: <matplotlib.image.AxesImage at 0x7f624947d630>



In [0]:
```
new_target
```

Out[0]: 9.0

The Competition:

- Lasted roughly 3 days
- We explored a variety of papers and combining various ideas to see what *together* could work the best

Papers Referenced:

- [Bag of Tricks for Resnet (aka the birth of xResNet) (https://arxiv.org/abs/1812.01187)](https://arxiv.org/abs/1812.01187)
- [Large Batch Optimization for Deep Learning, LAMB (https://arxiv.org/abs/1904.00962)](https://arxiv.org/abs/1904.00962)
- [Large Batch Training of Convolutional Networks, LARS (https://arxiv.org/pdf/1708.03888.pdf)](https://arxiv.org/pdf/1708.03888.pdf)
- [Lookahead Optimizer: k steps forward, 1 step back (https://arxiv.org/abs/1907.08610)](https://arxiv.org/abs/1907.08610)
- [Mish: A Self Regularized Non-Monotonic Neural Activation Function (https://arxiv.org/abs/1908.08681v1)](https://arxiv.org/abs/1908.08681v1)
- [On the Variance of the Adaptive Learning Rate and Beyond, RAdam (https://arxiv.org/abs/1908.03265)](https://arxiv.org/abs/1908.03265)
- [Self-Attention Generative Adversarial Networks (https://arxiv.org/abs/1805.08318)](https://arxiv.org/abs/1805.08318)
- Stochastic Gradient Methods with Layer-wise Adaptive

Moments for Training of Deep Networks, Novograd (https://arxiv.org/pdf/1905.11286.pdf)

Other Equally as Important Noteables:

- Flatten + Anneal Scheduling - Mikhail Grankin
- Simple Self Attention - Seb

One trend you will notice throughout this exercise is we (everyone mentioned above and myself) all tried combining a variety of these tools and papers together before Seb eventually came up with the winning solution. For a bit of context, here is the pre-competition State of the Art for ImageWoof:

Imagewoof

Size (px)	Epochs	Accuracy	URL	Params
128	5	55.2	link	`--epochs 5 --bs 64 --lr 3e-3 --mixup 0 --woof 1`

And here was the winning results:

20 total	5 epoch runs:	Results:		Gains:	
0.742	0.752	**74.97%**	Average	**19.77%**	vs Leaderboard
0.78	0.768	**78.00%**	Max	**22.80%**	vs Leaderboard
0.756	0.75	0.0130	SDev		(55.2% current)
0.77	0.756				
0.732	0.732				
0.76	0.748				
0.742	0.754				
0.736	0.752				
0.736	0.74				
0.738	0.75				

As a general rule of thumb, we always want to make sure our results are reproducable, hence the multiple runs and reports of the Standard Deviation, Mean, and the Maximum found. For today, we will just do one run of five for time. Following no particular order, here is a list of what was tested, and what we will be testing today:

- Baseline (Adam + xResnet50) + OneCycle
- Ranger (RAdam + LookAhead) + OneCycle
- Ranger + Flatten Anneal
- Ranger + MXResnet (xResnet50 + Mish) + Flatten Anneal
- RangerLars (Ralamb + LARS + Ranger) + Flatten Anneal
- RangerLars + xResnet50 + Flatten Anneal
- Ranger + SimpleSelfAttention + MXResnet + Flatten Anneal

The last of which did achieve the best score overall.

Functions:

Functions:

For the sake of simplicity, we will borrow from Seb's gitub repository.

In [0]:
```
!git clone https://github.com/sdoria/mish
```

```
Cloning into 'mish'...
remote: Enumerating objects: 46, done.
remote: Counting objects: 100% (46/46), don
e.
remote: Compressing objects: 100% (40/40),
done.
remote: Total 46 (delta 21), reused 19 (del
ta 6), pack-reused 0
Unpacking objects: 100% (46/46), done.
```

In [0]:
```
%cd mish
from rangerlars import *
from mish import *
from mxresnet import *
from ranger import *
```

```
/content/mish
Mish activation loaded...
```

Running the tests

For our tests, we will use the overall accuracy as well as the top_k, as this is what was used in Jeremy's example. Do note that top_k is not quite as relevent here as we only have 10 classes

Baseline

In [0]:
```
opt_func = partial(optim.Adam, betas=(0.9,
0.99), eps=1e-6)
```

In [0]:
```
learn = Learner(data, models.xresnet50(c_ou
t=10), wd=1e-2, opt_func=opt_func,
               bn_wd=False, true_wd=True, l
oss_func=LabelSmoothingCrossEntropy(),
               metrics=[accuracy, top_k_acc
uracy])
```

In [0]:
```
learn.fit_one_cycle(5, 3e-3, div_factor=10,
pct_start=0.3)
```

epoch	train_loss	valid_loss	accuracy	top_k_acc
0	2.153507	2.155606	0.236000	0.764000
1	1.950127	2.465281	0.282000	0.788000
2	1.722233	1.586035	0.488000	0.932000

3	1.523372	1.403256	0.588000	0.952000
4	1.379958	1.315990	0.624000	0.970000

Ranger + OneCycle

In [0]:
```
opt_func = partial(Ranger, betas=(0.9,0.99
), eps=1e-6)
```

In [0]:
```
learn = Learner(data, models.xresnet50(c_ou
t=10), wd=1e-2, opt_func=opt_func,
                bn_wd=False, true_wd=True, l
oss_func=LabelSmoothingCrossEntropy(),
                metrics=[accuracy, top_k_acc
uracy])
```

In [0]:
```
learn.fit_one_cycle(5, 3e-3, div_factor=10,
pct_start=0.3)
```

epoch	train_loss	valid_loss	accuracy	top_k_acc
0	1.897928	2.008069	0.312000	0.796000
1	1.791323	1.728758	0.418000	0.898000
2	1.669062	1.666615	0.478000	0.912000
3	1.570262	1.517981	0.548000	0.936000
4	1.525395	1.487397	0.548000	0.938000

Ranger + Flatten Anneal

In [0]:
```
opt_func = partial(Ranger, betas=(0.9,0.99
), eps=1e-6)
learn = Learner(data, models.xresnet50(c_ou
t=10), wd=1e-2, opt_func=opt_func,
                bn_wd=False, true_wd=True, l
oss_func=LabelSmoothingCrossEntropy(),
                metrics=[accuracy, top_k_acc
uracy])
```

In [0]:
```
learn.fit_fc(5, 3e-3)
```

epoch	train_loss	valid_loss	accuracy	top_k_acc
0	2.098281	2.399817	0.266000	0.744000
1	1.929641	2.321711	0.302000	0.798000
2	1.733089	1.623181	0.506000	0.920000

3	1.582382	1.617398	0.496000	0.924000
4	1.361795	1.311129	0.670000	0.952000

Ranger + MXResnet + Flatten Anneal

In [0]:
```
opt_func = partial(Ranger, betas=(0.9,0.99
), eps=1e-6)
learn = Learner(data, mxresnet50(c_out=10),
wd=1e-2, opt_func=opt_func,
               bn_wd=False, true_wd=True, l
oss_func=LabelSmoothingCrossEntropy(),
               metrics=[accuracy, top_k_acc
uracy])
```

In [0]:
```
learn.fit_fc(5, 4e-3)
```

epoch	train_loss	valid_loss	accuracy	top_k_acc
0	2.054917	2.199414	0.286000	0.804000
1	1.804237	3.025912	0.254000	0.732000
2	1.616225	1.517143	0.574000	0.944000
3	1.449524	1.379319	0.622000	0.938000
4	1.221281	1.168319	0.728000	0.958000

RangerLars + MXResnet + Flatten Anneal

In [0]:
```
opt_func = partial(RangerLars, betas=(0.9,
0.99), eps=1e-6)
learn = Learner(data, mxresnet50(c_out=10),
wd=1e-2, opt_func=opt_func,
               bn_wd=False, true_wd=True, l
oss_func=LabelSmoothingCrossEntropy(),
               metrics=[accuracy, top_k_acc
uracy])
```

In [0]:
```
learn.fit_fc(5, 4e-3)
```

epoch	train_loss	valid_loss	accuracy	top_k_acc
0	1.945484	2.401585	0.318000	0.780000
1	1.714290	1.956744	0.384000	0.844000
2	1.587898	1.804619	0.416000	0.906000

3	1.502960	1.540900	0.536000	0.928000
4	1.361548	1.342270	0.646000	0.954000

RangerLars + xResnet50 + Flatten Anneal

In [0]:
```
opt_func = partial(RangerLars, betas=(0.9,
0.99), eps=1e-6)
learn = Learner(data, models.xresnet50(c_ou
t=10), wd=1e-2, opt_func=opt_func,
               bn_wd=False, true_wd=True, l
oss_func=LabelSmoothingCrossEntropy(),
               metrics=[accuracy, top_k_acc
uracy])
```

In [0]:
```
learn.fit_fc(5, 4e-3)
```

epoch	train_loss	valid_loss	accuracy	top_k_acc
0	2.006913	2.315927	0.284000	0.708000
1	1.801610	1.927570	0.378000	0.852000
2	1.703221	1.858955	0.394000	0.880000
3	1.643228	1.700991	0.448000	0.872000
4	1.488607	1.462179	0.594000	0.940000

Ranger + SimpleSelfAttention + MXResnet + Flatten Anneal

In [0]:
```
opt_func = partial(Ranger, betas=(0.95,0.99
), eps=1e-6)
learn = Learner(data, mxresnet50(c_out=10,
sa=True), wd=1e-2, opt_func=opt_func,
               bn_wd=False, true_wd=True, l
oss_func=LabelSmoothingCrossEntropy(),
               metrics=[accuracy, top_k_acc
uracy])
```

In [0]:
```
learn.fit_fc(5, 4e-3)
```

epoch	train_loss	valid_loss	accuracy	top_k_acc
0	1.968602	2.051655	0.330000	0.830000
1	1.709247	2.174664	0.384000	0.892000
2	1.537062	1.451682	0.598000	0.946000

3	1.393161	1.419797	0.578000	0.954000
4	1.172150	1.122868	0.746000	0.978000

As we can see, 74.6 is what we got. The highest recorded is 78%.

From here:

I encourage you all to try out some of the combinations seen here today and apply a bit more to it. For instance, are we using the best hyperparameters? What about Cut-Out? MixUp? Plenty more to explore!

ClassConfusion

Lastly is ClassConfusion. This is meant to help explain how your model is behaving and understand where it's weaknesses are. We will examine it through images today, and we will look at it for tabular next week.

docs (https://docs.fast.ai/widgets.class_confusion.html)

For use with regular jupyter notebooks, use `from fastai.widgets import ClassConfusion`

For use with Google Colab, use my repo: repo (https://github.com/muellerzr/ClassConfusion)

In [0]:
```
!git clone https://github.com/muellerzr/ClassConfusion
```

```
Cloning into 'ClassConfusion'...
remote: Enumerating objects: 50, done.
remote: Counting objects: 100% (50/50), done.
remote: Compressing objects: 100% (50/50), done.
remote: Total 334 (delta 28), reused 0 (delta 0), pack-reused 284
Receiving objects: 100% (334/334), 2.13 MiB | 13.21 MiB/s, done.
Resolving deltas: 100% (194/194), done.
```

In [0]:
```
from ClassConfusion import *
```

In [0]:
```
interp = ClassificationInterpretation.from_learner(learn)
```

In [0]:
```
interp.most_confused()[:5]
```

Out[0]:
```
[('n02089973', 'n02088364', 20),
 ('n02096294', 'n02087394', 14),
```

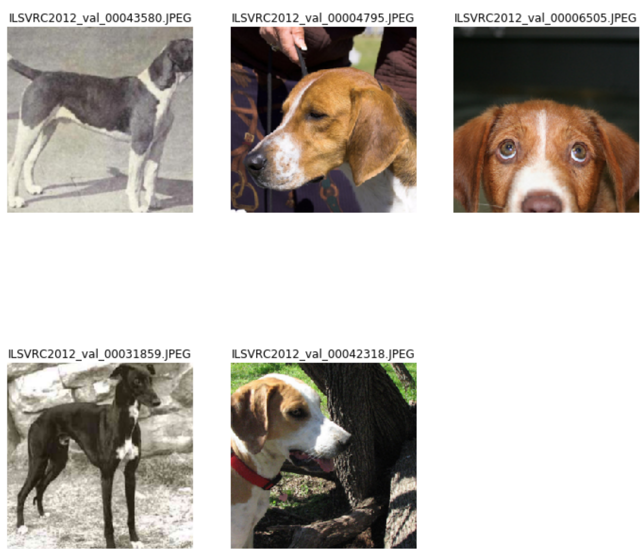
```
 ('n02099601', 'n02087394', 14),
 ('n02096294', 'n02093754', 10),
 ('n02105641', 'n02086240', 10)]
```

In [0]:
```
comboList = [('n02089973', 'n02088364')]
```

In [0]:
```
ClassConfusion(interp, comboList, is_ordered=True, figsize=(12,12))
```

```
Please enter a value for `k`, or the top images you will see: 5
```

```
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
  0%|          | 0/1 [00:00<?, ?it/s]
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
<IPython.core.display.Javascript object>
```




```
<Figure size 432x288 with 0 Axes>
<IPython.core.display.Javascript object>
100%|██████████| 1/1 [00:01<00:00,  1.62s/it]
```

Out[0]:
```
<ClassConfusion.classConfusion.ClassConfusion at 0x7f3aa02f9438>
```

In [0]:
```
comboList = ['n02089973', 'n02088364', ]
```